Simple Web Services
Internet Basics

- The Internet is based on a communication protocol named TCP (Transmission Control Protocol)
- TCP allows programs running on different computers to connect and communicate directly with each other
- TCP requires that each computer have a unique identifier called an “IP Address”
  - 128.187.80.20
  - 72.30.38.140
Internet Basics

• Since a computer runs many programs simultaneously, TCP uses Port Numbers to identify individual programs running on a computer
  – TCP Port Numbers are in the range 0 – 65535
  – Ports 0 – 1023 are reserved for system services (email, web, etc.)
  – Ports 1024 – 49151 are registered to particular applications
  – Ports 49152 – 65535 can be used for custom or temporary purposes
  – Email servers typically run on Port 25
  – Web servers typically run on Port 80
Internet Basics

• The combination of (IP Address, TCP Port Number) uniquely identifies a particular program on a particular computer
  – (128.187.80.20, 25) => Email server on machine 128.187.80.20
  – (72.30.38.140, 80) => Web server on machine 72.30.38.140
Internet Basics

• Through TCP, a program on one computer can connect to a program running on another computer by specifying its (IP Address, TCP Port Number)
  – Connect to (128.187.80.20, 25) => Connect to email server on machine 128.187.80.20
  – Connect to (72.30.38.140, 80) => Connect to web server on machine 72.30.38.140

• Such a TCP connection is called a “Socket”

• Once a connection has been established, the two programs can pass data back and forth to each other (i.e., communicate)
Internet Basics

• IP Addresses are hard to remember and work with directly

• Users prefer to reference machines by Name rather than by IP Address
  – pinky.cs.byu.edu instead of 128.187.80.20
  – www.yahoo.com instead of 72.30.38.140

• DNS (Domain Name System) is a protocol for looking up a machine’s IP Address based on its (Domain) Name
  – Connect to (www.yahoo.com, 80)
  – DNS, what is the IP Address for “www.yahoo.com”?
  – 72.30.38.140
  – OK, Connect to (72.30.38.140, 80)
URLs (uniform resource locators)

scheme://domain:port/path?query_string#fragment_id

- **scheme** (case-insensitive) – http or https
- **domain** (case-insensitive) – The server’s domain name or IP address. The domain name google.com, or its IP address 72.14.207.99, is the address of Google's website.
- **port** (optional) – The port, if present, specifies the server’s TCP port number. For http URLs, the default port is 80. For https URLs, the default port is 443.
- **path** (case-sensitive) – The path is used to specify and perhaps locate the requested resource.
- **query_string** (optional, case-sensitive) – The query string, if present, contains data to be passed to software running on the server. It may contain name/value pairs separated by ampersands, for example ?first_name=John&last_name=Doe.
- **fragment_id** (optional, case-sensitive) – The fragment identifier, if present, specifies a part or a position within the overall resource or document.
URLs

http://www.espn.com:80/basketball/nba/index.html?team=dallas&order=name#Roster

- **scheme** – http
- **domain** – www.espn.com
- **port** – 80
- **path** – /basketball/nba/index.html
- **query_string** – ?team=dallas&order=name
- **fragment_id** – #Roster
Java’s URL Class

```java
import java.net.URL;

URL url = new URL(
"http://www.espn.com:80/basketball/nba/index.html?
team=dallas&order=name#Roster");

String host = url.getHost();
int port = url.getPort();
String path = url.getPath();
String query = url.getQuery();
String fragment = url.getRef();

// Many more URL operations
```
HTTP
(hypertext transfer protocol)

• Network protocol that drives the Web

• Built on top of TCP

• By default, Web servers run on TCP Port 80

• HTTP has a Request/Response structure
  – Client (e.g., web browser) sends a “request” message to the server
  – Server sends back a “response” message to the client
HTTP Request message format

```
<method>  <request-URL>  <version>\r\n.Annotations[headers]\r\n\r\n<entity-body>
```

<method> is the operation to perform on URL
<request-URL> can be full URL or just the path part
<version> is of the form HTTP/<major>.<minor>
<entity-body> is a stream of bytes (could be empty)

GET /test/hi-there.txt HTTP/1.1
Accept: text/*
Host: www.joes-hardware.com
HTTP Response message format

<version>  <status>  <reason-phrase>\r\n<headers>\r\n\r\n<entity-body>

<version> is of the form HTTP/<major>.<minor>
$status$ is a 3-digit number indicating status of request
(reason-phrase) human-readable description of status code
(entity-body) is a stream of bytes (could be empty)

HTTP/1.0 200 OK
Content-type: text/plain
Content-length: 18

Hi! I’m a message!
HTTP Request Methods

• **GET** – Retrieve document from server
• **POST** – Send data to server for processing
  • PUT – Store document on server
  • DELETE – Remove document from server
  • HEAD – Retrieve document headers from server
  • OPTIONS – Determine what methods the server supports
  • TRACE – Trace the path taken by a request through proxy servers on the way to the destination server
HTTP Response status codes

- 100-199  Informational
- 200-299  Successful
- 300-399  Redirection
- 400-499  Client error
- 500-599  Server error

- 200  OK
- 401  Unauthorized to access resource
- 404  Requested resource does not exist
HTTP Headers

• List of name/value pairs
• Name: Value\r\n• Empty line separates headers and entity body

• General headers (request or response)
  - Date: Tue, 3 Oct 1974 02:16:00 GMT
    • Time at which message was generated
  
  - Connection: close
    • Client or server can specify options about the underlying connection
HTTP Request Headers

- **Host**: www.joes-hardware.com
  - Host from the request URL

- **User-Agent**: Mozilla/4.0
  - Client application making the request

- **Accept**: text/html, text/xml
  - MIME types the client can handle

  - Page that contained the link currently being requested

- **If-Modified-Since**: Tue, 3 Oct 1974 02:16:00 GMT
  - Conditional request; only send the document if it changed since I last retrieved it
HTTP Response Headers

- **Content-length**: 15023
  - Length of response entity body measured in bytes

- **Content-type**: text/html
  - MIME type of response entity body

- **Server**: Apache/1.2b6
  - Server software that handled the request

- **Cache-Control**: no-cache
  - Clients must not cache the response document
Web Access in Java

• Java’s `URLConnection` class can be used by clients to make HTTP requests and receive HTTP responses
Java’s URLConnection class (GET)

try {
    URL url = new URL("http://www.byu.edu/");

    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    connection.setRequestMethod("GET");
    // Set HTTP request headers, if necessary
    // connection.addRequestProperty("Accept", "text/html");
    connection.connect();

    if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
        // Get HTTP response headers, if necessary
        // Map<String, List<String>> headers = connection.getHeaderFields();

        // Get response body input stream
        InputStream responseBody = connection.getInputStream();

        // Read response body bytes
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int length = 0;
        while ((length = responseBody.read(buffer)) != -1) {
            baos.write(buffer, 0, length);
        }
    }

    // Convert response body bytes to a string
}

Java’s URLConnection class (POST)

String postData = "Whatever string you want to send to the server";
try {
    URL url = new URL("http://www.byu.edu/");

    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

    connection.setRequestMethod("POST");
    connection.setDoOutput(true);

    // Set HTTP request headers, if necessary
    // connection.addRequestProperty("Accept", "text/html");

    connection.connect();

    // Write post data to request body
    OutputStream requestBody = connection.getOutputStream();
    requestBody.write(postData.getBytes());
    requestBody.close();

    if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
        // Get HTTP response headers, if necessary
        // Map<String, List<String>> headers = connection.getHeaderFields();

        // Get response body input stream
        InputStream responseBody = connection.getInputStream();

        // Read response body bytes
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
    }
}
Family Map Server

- Server web service interface (or API)
  - Login
  - Get People
  - Get Events
Family Map Server

- Server web service interface (or API)
  - Operation: Login
    - HTTP Request
      - URL: http://host:port/user/login
      - Type: POST
      - Headers: None
      - Body: `{ username: "username", password: "password" }`
    - HTTP Response
      - On Success
        » Status code: 200 (OK)
        » Body: {
          "Authorization": "auth-token",
          "userName": "username",
          "personId": "user’s person id"
        }
      - On Failure
        » Status code: 4xx or 5xx
        » Body: None
Family Map Server

- Server web service interface (or API)
  - Operation: Get All People
    - HTTP Request
      - URL: http://host:port/person/
      - Type: GET
      - Headers: Authorization: auth-token
      - Body: None
    - HTTP Response
      - On Success
        » Status code: 200 (OK)
        » Body: JSON person data
      - On Failure
        » Status code: 4xx or 5xx
        » Body: None
Family Map Server

- Server web service interface (or API)
  - Operation: Get All Events
    - HTTP Request
      - URL: http://host:port/event/
      - Type: GET
      - Headers: Authorization: auth-token
      - Body: None
    - HTTP Response
      - On Success
        » Status code: 200 (OK)
        » Body: JSON event data
      - On Failure
        » Status code: 4xx or 5xx
        » Body: None